Behavioral to RTL Coding
Objectives

After completing this module, you will be able to...

- Examine the two primary styles of writing VHDL code
  - Consider the differences and applications
  - Topics Include
    - Synthesis Vs. Simulation
    - Defining Behavioral
    - Defining RTL
    - Source Code Portability
    - Modeling DFFs
    - State Machines
Levels of Abstraction

Fewer details, Faster design entry and simulation

Behavioral

RTL

Logic

Layout

Technology specific details, slower design entry and simulation
A given module may be written in a Behavioral or RTL style depending on the intent. Specifically, what is the model to be used for at this time?
Behavorial Coding

- VHDL is a Hardware Modeling Language!
- Some of the considerable features and flexibility that support behavioral modeling and verification include:
  - Assignments are clean
  - Values stored in signal
  - Model bus input / output
  - No need to consider propagation delay
  - Model timing as necessary
  - Arbitrary sequencing of events
  - Zero execution time
  - Delta-cycle between simulation time

```vhdl
Z <= A or B after 2 ns;

wait for 10 ns;

Z <= A or B;

A_bus <= "1010";

signal A_bus : bit_vector (3 downto 0) := "0011";
```
entity Security_1 is
port (Clk, Reset : in std_logic ;
       Keypad : in std_logic_vector (3 downto 0) ;
       Front_Door, Rear_Door, Window : in boolean ;
       Alarm_Siren : out boolean ) ;
end Security_1 ;

architecture Behave of Security_1 is
constant Delay_Period : time := 20 s ;
begin
  process ( Keypad, Front_Door, Rear_Door, Window )
  begin
    if (Front_Door or Rear_Door or Window ) then
      if (Keypad = "0011") then
        Alarm_Siren <= false ;
      else
        Alarm_Siren <= true after Delay_Period ;
      end if ;
    end if ;
  end process ;
end Behave ;
RTL Coding

Coding at the RTL (Register Transfer Level) requires more detailed description, with synthesizable constructs only. RTL level code is inherently synchronous as we describe the transformations (logic) between clock-edges (registers)

Model register operation
Asynchronous initialization conditions
Synchronous initialization conditions
Consider propagation delays
Clocked processes
Combinatorial processes
Portability

Combinatorial process

Clocked process
For synthesis of combinatorial logic, all signals “read” in the process must be included in the process sensitivity list. A signal is considered to “read” if its value must be determined in the execution of the sequential statement.

```
process (A,B)
begin
    Z <= A or B;
end process;
```

**What is the behavior (simulation) if “B” is not in the sensitivity list?**
Inferring Latches

- For combinatorial logic described with an “if” statement, default assignments are necessary to prevent latch inference in logic synthesis
  1) Also, assign to ALL signals in every branch of if/then & case

```
process ( D, En )
begin
  if En = '1' then
    Q <= D ;
  end if ;
end process ;
```

What happens when En /=‘1’?
The current value of Q will need to be maintained

Why is this not an issue for combinatorial logic described with a “case” statement?
Inferring Registers

- All signals assigned within a clocked process will be registered. Consequently, you should not assign a signal within a clocked process unless you intend to register it.

```vhdl
process ( Clk, Reset )
begin
if Reset = '1' then
    Q <= '0';
    Z <= '0';
    M <= '0';
    Y <= '0';
elsif (Clk'event and Clk = '1') then
    Q <= D ;
    Z <= A ;
    M <= Z ;
    Y <= X;
end if ;
end process ;
```
Detecting Rising Edges

- For synthesis of registered logic (clocked processes), we start by detecting the rising or falling edge of the signal that will trigger the register. It's important to follow industry standards to maintain the utmost source code portability.

```vhdl
process (Clk, reset)
begin
  if reset = '1' then
    Q <= '0';
  elsif (Clk`event and Clk = '1') then
    Q <= D;
  end if;
end process;
```

- Note that for a clocked process, only the clock signal and reset need be in the sensitivity list, not the data inputs to the register.

- The `event` attribute is a VHDL signal attribute that returns a Boolean “true” if there was a event (change) on that signal during the current simulation cycle.
Rising Edge Function Call

- The VHDL Std_logic_1164 package provides a simple function call (sub-program) for the detection of a rising edge

  — There is a corresponding utility called “falling edge”

```vhdl
process ( Clk, Reset )
begin
  if Reset = '1' then
    Q <= '0';
  elsif rising_edge (Clk) then
    Q <= D;
  end if;
end process;
```

```vhdl
package std_logic_1164 is
  . . .
function rising_edge ( signal Clk : std_logic ) return boolean;
  begin
    if ( Clk'event and Clk= '1' and Clk'last_value = '0' ) then
      return true;
    else
      return false;
    end if;
end rising_edge;
  . . .
```
```vhdl
architecture RTL of Security_1 is
  type Alarm_State is (Waiting, Armed, Dis-Armed);
  signal State : Alarm_State;
  begin
    State <= Dis-Armed;
    Alarm_Siren <= false;
    elsif (Clk'event and Clk = '1') then
      case State is
        when Waiting =>
          if Keypad = "0011" then
            State <= Dis-Armed;
            Alarm_Siren <= false;
          elsif (Clk'event and Clk = '1') then
            case State is
              when Waiting =>
                if Keypad = "0011" then
                  State <= Dis-Armed;
                  Alarm_Siren <= false;
                else State <= Armed;
                end if;
              when Armed =>
                if (Front_Door or Rear_Door or Window) then
                  if Keypad /= "0011" then
                    Alarm_Siren <= true;
                  else State <= Waiting;
                  end if;
                end if;
              when Dis-Armed =>
                Alarm_Siren <= false;
                State <= Waiting;
                end if;
              end case;
            end if;
          end elsif;
        end case;
    end process;
end architecture RTL;
```
Two Process State-Machine

entity Security_1 is
  port (Clk, Reset : in std_logic ;
       Keypad : in std_logic_vector (3 downto 0) ;
       Front_Door, Rear_Door, Window : in boolean ;
       Alarm_Siren : out boolean ) ;
end entity Security_1 ;

architecture RTL of Security_1 is
  type Alarm_State is (Waiting, Armed, Dis-Armed ) ;
  signal Current_State, Next_State : Alarm_State ;
begin

  Recommended!

  Sync: process ( Clk, Reset )
  begin
    if (Reset = '1') then
      Current_State <= Dis-Armed ;
    elsif ( Clk'event and Clk = '1') then
      Current_State <= Next_State ;
  end process Sync ;

  Comb: process (Current_State, Front_Door, Rear_Door, Window, Keypad)
  begin
    case (Current_State) is
      when Waiting =>
        if Keypad = "0011" then
          Next_State <= Dis-Armed ;
        else
          Next_State <= Armed ;
        end if ;
      when Armed =>
        if ( Front_Door or Rear_Door or Window ) then
          if Keypad /= "0011" then
            Alarm_Siren <= true ;
          else
            Next_State <= Waiting ;
          end if ;
        end if ;
      when Dis-armed =>
        Alarm_Siren <= false ;
        Next_State <= Waiting ;
      end case ;
    end if ;
  end process Comb ;
end architecture RTL ;

Summary

- Behavioral constructs allow for detailed modeling
- RTL code should contain only synthesizable constructs
- For combinatorial logic, all signals read in the process should be included in the sensitivity list
- For clocked processes, only the clocking signal and an asynchronous initialization-- if present, should be in the sensitivity list
- Avoid inadvertent latch inference
- Follow the rules, maintain the utmost code portability
Review Questions

- Which statement is not true for behavioral simulation?
  1. Values are stored in signals
  2. RTL statements are not allowed
  3. You can model bus I/O

- A VHDL synthesis compiler uses the process sensitivity list for?

- The `event` attribute tests for what?

- Using the rising_edge function call promotes:
  1. Readability
  2. RTL Code Portability
  3. Retention
  4. Faster Simulation
Review Answers

- Which statement is not true for behavioral simulation?
  2. RTL statements are not allowed (Any RTL construct can be behaviorally simulated)

- A VHDL synthesis compiler uses the process sensitivity list for?
  Nothing; only the behavioral simulator uses process sensitivity list

- The `event` attribute tests for what?
  Returns boolean true if the signal change occurred in the current simulation cycle

- Using the rising_edge function call promotes:
  2. RTL Code Portability (The function call was created for that purpose, as well as for readability)